Yao, Zhen; Yan, Zheng

A trust management framework for software-defined network applications

Published in:
Concurrency and Computation: Practice and Experience

DOI:
10.1002/cpe.4518

Published: 25/08/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Yao, Z., & Yan, Z. (2020). A trust management framework for software-defined network applications. *Concurrency and Computation: Practice and Experience*, *32*(16), Article e4518. https://doi.org/10.1002/cpe.4518

A Trust Management Framework for Software-Defined Network Applications

Zhen Yao[1], Zheng Yan[1,2*]

[1]The State Key Laboratory on Integrated Services Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China

[2]Department of Communications and Networking, Aalto University, Espoo 02150, Finland

*Corresponding author: Zheng Yan (zyan@xidian.edu.cn; zhengyan.pz@gmail.com)

Abstract

The emergence of Software-Defined Network (SDN) has brought unprecedented innovation to current networks. SDN's two most notable features are decoupling and programmability. Decoupling makes network management centralized in a control plane. Meanwhile, benefitted from the programmable characteristic of SDN, new functions of networking can be easily realized. However, these features also introduce new security issues to SDN. Through the programming interface provided by SDN, software engineers can easily develop network applications to generate networking policies for SDN's control planes for the purpose of guiding network routing. But it is hard to guarantee the security and quality of these new applications. Malicious or low-quality applications could damage a whole network. To solve this problem, we propose a novel trust management framework for SDN applications in this paper. It can evaluate applications' trust values based on their impact on the network performance (such as time delay, packet loss rate, throughput, etc.). These trust values further play a decisive role for managing and selecting applications in SDN. We evaluate this framework's performance through a prototype system implemented based on a floodlight controller. The experimental results show the accuracy and effectiveness of our design.

***Keywords**:* Software Defined Network, SDN Application Security, Trusted Computing Platform, Trust Management, Trust Evaluation.

1. Introduction

Traditional network architecture is a kind of distributed architecture, and the functions of control and data forwarding are distributed in each router. This kind of network architecture appears an increasing number of drawbacks. Above all, because of a distributed control mode, network maintenance and operation are very complicated for network administrators. Second, switching equipment of traditional networks integrates control function and forwarding function, so network upgrade needs to replace a large amount of underlying switching devices, which is a huge expense to telecommunication operators. What's more, in traditional networks, the deployment of new network functions requires network administrators to separately configure each

device, this kind of working mode is hard to meet the demand of fast development of telecommunication networks and Network Function Virtualization (NFV) [1].

Software-Defined Network (SDN) makes a great contribution to solve the above problems. SDN was born at the Stanford University in 2006. It was built to improve the management of Stanford campus networks [2]. Its core design idea is to decouple the data forwarding and routing control functions of traditional Internet Protocol networks to achieve centralized control, distributed forwarding and programmable characteristics. Network centralized management brings a global network view [3] to network administrators and makes network configuration no longer a huge workload. Meanwhile, network's update can be realized easily through SDN's Application Programming Interface (API). Once an application is adopted in the application plane of SDN, SDN controllers can translate its offered policies to flow rules and automatically distribute the rules to specific switches to implement specific functionalities. SDN has drawn wide attention from both academia and industry. Until now, it has become one of core technologies of 5G wireless communication networks [33], and the implementation of NFV can depend on SDN's architecture [37, 38]. Most of commercial switch vendors and telecommunications operators have begun to produce or deploy SDN products [4].

1.1 Motivation

As a new technology, SDN greatly contributes to network innovation, but it also brings new security challenges [5, 32]. As introduced above, SDN has two specific characteristics [31]. One is that the control plane of SDN is centrally responsible for the management of a whole network. The other is that the emergence of the application plane of SDN makes the network programmable. However, SDN's centralized control makes its controllers more vulnerable to Denial of Service attacks or Distributed Denial of Service attacks (DoS/DDoS) [6]. Current solutions to mitigate DoS/DDoS attacks mainly include DoS/DDoS detection [8-11], controller optimization [12, 13] and loading balance [14]. Because the controller holds a large amount of information about entire network structure, once it is invaded, the leakage of network information will cause a big harm [7].

In this paper, we focus on the security issue of the application plane. The SDN controllers can easily enable the 3rd party applications installed into the SDN architecture, but they have no ability to distinguish the applications' eligibility, legality and trustworthiness by themselves [5]. Besides that, a poorly designed or buggy application could unintentionally bring a series of vulnerabilities to the SDN system. In short, the application plane mainly faces three security problems: 1) Lack of certification and authorization: It lacks a robust authentication and authorization mechanism for applications, especially for a large number of third-party applications. 2) Fraud traffic insertion: Malicious or buggy applications could generate erroneous harmful flow rules and routing policies, but compromised applications are hard to be detected. 3) Conflicts between applications: each application generates lots of flow

rules, but it's hard to guarantee that there are no conflicts between these rules [40]. Therefore, how to deal with the conflicts among applications is another challenge of SDN security.

In order to solve the application plane's security issues, the controllers must have the ability to identify whether the applications are trustworthy, or there are some additional functions in SDN to assist the controllers to make correct justification on this. In the literature, some role-based application authorization methods were proposed to deal with application policy conflicts and to manage applications [15-18]. In [21, 22], authors focused on improving the resilience of controllers to both buggy and malicious applications. In [19-20, 23, 36, 39], authors added a new function into an SDN system to distribute privileges to SDN applications. However, the above methods are deficient in establishing trust relationships between applications and controllers. In particular, the role-based method is not fine-grained [15-18]. Applications' impact on the data plane of SDN was ignored in the past work [21, 22]. Therefore, how to reasonably evaluate SDN applications according to their actual impact on networks becomes a significant issue [41, 42]. Based on the evaluation on applications, it is possible and capable for a SDN system to distinguish applications and manage them.

1.2 Main Contributions

In this paper, we propose a new Trust Management Framework (TMF) for SDN applications. It evaluates each application's trust value based on its performance. This trust value can be used to solve applications' conflicts and detect malicious applications. Our system mainly contains two modules: Network Performance Monitor (NPM) module and Trust Evaluation (TE) module. The NPM module consists of a number of probes that are configured to monitor the performance of flow rules issued by different applications when they are fulfilled in the data plane. These monitoring results will be sent to TE module with corresponding tags that indicate the applications that cause the represented network performance and the time when the network is affected. After collecting the above data and the feedback from other application users (i.e., other controllers), the TE module calculates the application's current trust value and sends the trust value to the control plane and the application plane. The controllers manage the applications based on their trust values. Meanwhile, a Conflict Detection (CD) module is newly embedded into the control plane to detect whether a new flow rule issued by an application conflicts with existing policies. In order to guarantee the robustness and trustworthiness of this system, we introduce Trusted Platform Module (TPM) [29] into our proposed framework. TPM is applied to guarantee trust relationships among different modules in the system. In particular, we apply a trust sustainment and control mechanism based on TPM, e.g., a protocol as described in [28, 29], to ensure trustworthy performance monitoring and data collection for the purpose of high-quality trust evaluation. This protocol is designed to verify and further sustain the trust relationships among devices by attesting the device configurations and ensuring expected configurations are applied during device

cooperation by embedding trust conditions into TPM. Furthermore, we develop a prototype system using a floodlight controller based on the framework design. The performance evaluation results demonstrate the accuracy and effectiveness of our system.

The superiority of the proposed trust management framework system can be summarized as follows: 1) High Accuracy and Fine-graininess: For some applications with the same functions but different performance, the trust value evaluated by our system can well reflect their performance differences. 2): Dynamic and Real-time: the application's trust value evaluated by the TMF is not static, but changed according to its real-time impact on the network performance, which can be used to figure out malicious or forged applications. 3) Robustness and Trustworthiness: Due to the adoption of a trust sustainment and control mechanism based on TPM that is embedded into the NPM probes and the NPM module, the network performance data can be guaranteed as trusted, and further the trustworthiness of trust evaluation on each application can be ensured. Specifically, the contribution of this paper can be summarized as below:

- We propose a trust management framework for SDN applications that can evaluate application trust in order to mitigate application policy conflicts and detect malicious applications.
- We implemented a prototype system and design a series of experiments to test the accuracy and the efficiency of our framework.

The rest of this paper is organized as follows. In Section 2, we briefly review related work. Then, we introduce the preliminary knowledge about our proposed TMF in Section 3. The TMF design including system architecture and technical details is described in Section 4, followed by performance evaluation in Section 5. Finally, a conclusion is summarized in the last section.

2. Related Work

The programmable feature makes SDN face to a new challenge on how to manage the network applications to avoid maliciously utilizing this specific feature. In this section, we review related work about SDN application conflict detection and resolution and SDN application management.

2.1 SDN Application Conflict Detection and Resolution

SDN applications generate lots of flow rules when they are installed into controllers, and flow rule conflicts are inevitable. In this part, we review existing work related to application policy conflict detection and resolution. FortNox [15] is the software extension of a Nox controller (a kind of SDN controller) that provides flow rule conflict resolution based on the roles of applications. FortNox provides an algorithm to check flow rule contradictions in real time. When FortNox's rule conflict detection engine finds some conflicts between different applications, it chooses the flow rules produced by the application that has a higher security level set by the FortNox system.

Digital signature is used in FortNox to check applications' security levels. Three application roles are defined in this system with different security levels from high to low: administrators, security-related Openflow applications, and non-security-related Openflow applications. Based on the above work, the FortNox team also developed another security system called SE-floodlight [16], which was designed to solve the security issue of floodlight controller's applications. Similarly, the way that SE-floodlight resolves application conflicts is also to choose the application with a higher security role level. Obviously, the above methods are not fine-grained.

FRESCO [17] performs as a secure application development platform by combining with a NOX OPENFLOW controller. The basic framework of FRESCO consists of an application layer and a security enforcement kernel. The application layer provides four main functions: script-to-module translation, database management, event management and instance execution. The emergency of FRESCO makes it possible for a network manager to design or quickly develop a security application with a script language. In this work, they also use the method proposed in [15, 16] to mitigate rule conflicts, which is obviously not fine-grained.

In [18], the authors made efforts to solve policy conflicts between applications. They implemented a fully-functioned SDN controller called PANE that allows a network's administrator to safely delegate his authority using its APIs. Furthermore, they proposed a new algorithm for consolidating hierarchical policies and utilized this algorithm to accomplish application policies conflict resolution. This work takes advantage of policy atom to run a policy in an isolated environment. They used a tree structure to store flow rules. The nodes in the tree store the routing information of flow rules (e.g., IP number, port number, protocol, etc.). Its root node stores the flow rules' instructions. When a new policy is coming, the controller detects whether there are new conflicts by checking each node in the existing policy tree (from leaf nodes to root node). In this work, policy conflicts are resolved by dividing the usage level of applications.

In the above existing work, the majority methods to solve flow rule conflicts are based on the roles of applications that are distributed by their proposed system or network administrators. To a certain extent, the role-based method can solve the conflicts between applications, but it is not fine-grained. For example, when the roles of conflicting applications are same, which application's flow rules should be selected becomes a new problem.

2.2 SDN Application Management

We review some papers related to SDN application management herein. Hayward et al. designed a scheme to allocate permissions to network applications, which sets limitations on application operations [19]. They defined a set of permissions to which applications must subscribe during initialization with controllers and introduced an Operation Checkpoint that implements permission check prior to authorizing

application commands. This work's main contribution is to provide a method for network administrators to add, remove, change and query application permissions.

Christian et al. presented a web-based northbound interface, which is secure, controller independent, and supports the deployment of external applications [20]. In their work, an encrypted channel is used to communicate between SDN applications and controller. Meanwhile, they also proposed a trust management and resource-based access control model for SDN applications. They introduced a certificate authority to distribute privilege to SDN applications and managed them based on their certificates.

PermOF, a fine-grained permission system was presented in [23], which applies minimum privilege on applications. This system mainly considers two aspects: the most effective set of permissions and an isolation mechanism deployed to enforce permission control. It gives an action permission classification to each application and provides an isolation mechanism to enforce the permissions at an API entry.

Wang et al. proposed a permission management and authentication scheme called PERM-GUARD for SDN applications [36]. It employs a permission authentication model and introduces an identity-based signature scheme for the controller to verify the validity of applications' flow rules. In this work, they defined 16 kinds of permissions for each application. In [39], Wu et al. presented an access control model named Access Control Protector (AC-PROT) for SDN applications . AC-PROT employs an attribute-based signature scheme for SDN applications and defines 16 kinds of privilege levels for SDN applications. In the above methods [19-20, 23, 36, 39], the application's privilege is distributed by network administrators, which is different from our work presented in this paper. We manage applications according to their trust values that are evaluated based on application impact on networks.

Chandrasekaran et al. re-designed the controller architecture of SDN to make the controllers and the network resilient to application failures [21]. They presented LegoSDN that embodies described functions by providing two techniques. The first one is AppVisor – an isolation technique used in Operating Systems to separate address space of SDN applications from each other and controllers. The second one is NetLog – a network-wide transaction system that supports atomic updates and efficient roll backs. In this architecture, each application is run in an isolated Java Virtual machine (JVM), and it is handled by NETLog intensively. Once there are crashes in the applications, the NETLog will support the whole network back to a normal work status based on concerned roll-back strategies. This work focused on how to mitigate application failures' impact on networks. But it did not propose a detailed policy to manage applications.

Shin et al. provided a new controller called ROSEMARY [22], which has high resistance to malicious or buggy applications. They designed a micro network operating system called micro-NOS architecture. In this system, each application runs

in a sandbox. The system distributes a specific privilege to each application based on monitored resource consumption of the controller caused by applying the application. This work is advanced, but the authors focused on the influence of application on the control plane and ignored their impact on the data plane. Although some applications are less detrimental to the controller, they may cause paralysis of the underlying data plane.

We list application management methods and their management basis together with ours in [19-20, 22-23, 36, 39] in Table 1. To sum up, the existing work related to SDN application management focuses on applications' impact on controllers, and the management of applications mostly depends on third-party authorization. But how to reasonably determine applications' privilege level and how to detect the application that has a negative effect on the data plane in real time have not been well investigated in all above studies. In this paper, we attempt to propose a trust management framework to evaluate the trust of SDN applications in order to solve the above open issues.

Table 1. Comparison of Application Management Methods and Management Basis

Scheme	**Methods**	**Basis**
OPCheckpoint [19]	Permission List	15 kinds of permissions for each application
Secure-North [20]	Application access control	Application's certificates,
RoseMary [22]	Role-based	3 kinds of application authorization roles
PermOF [23]	Privilege List	18 kinds of privileges for each application
PERM-GUARD [36]	Identity-based	16 kinds of identities for each application
AC-PROT [39]	Privilege List	16 kinds of privileges for each application
Our Scheme	Trust-Based	Trust Value

3. Background Knowledge and Preliminaries

In this section, we introduce the background knowledge of this paper work. We firstly describe the structure of SDN and Openflow's flow table. Then, we introduce the trusted computing platform to explain the mechanism of TPM.

3.1 SDN Structure and Openflow

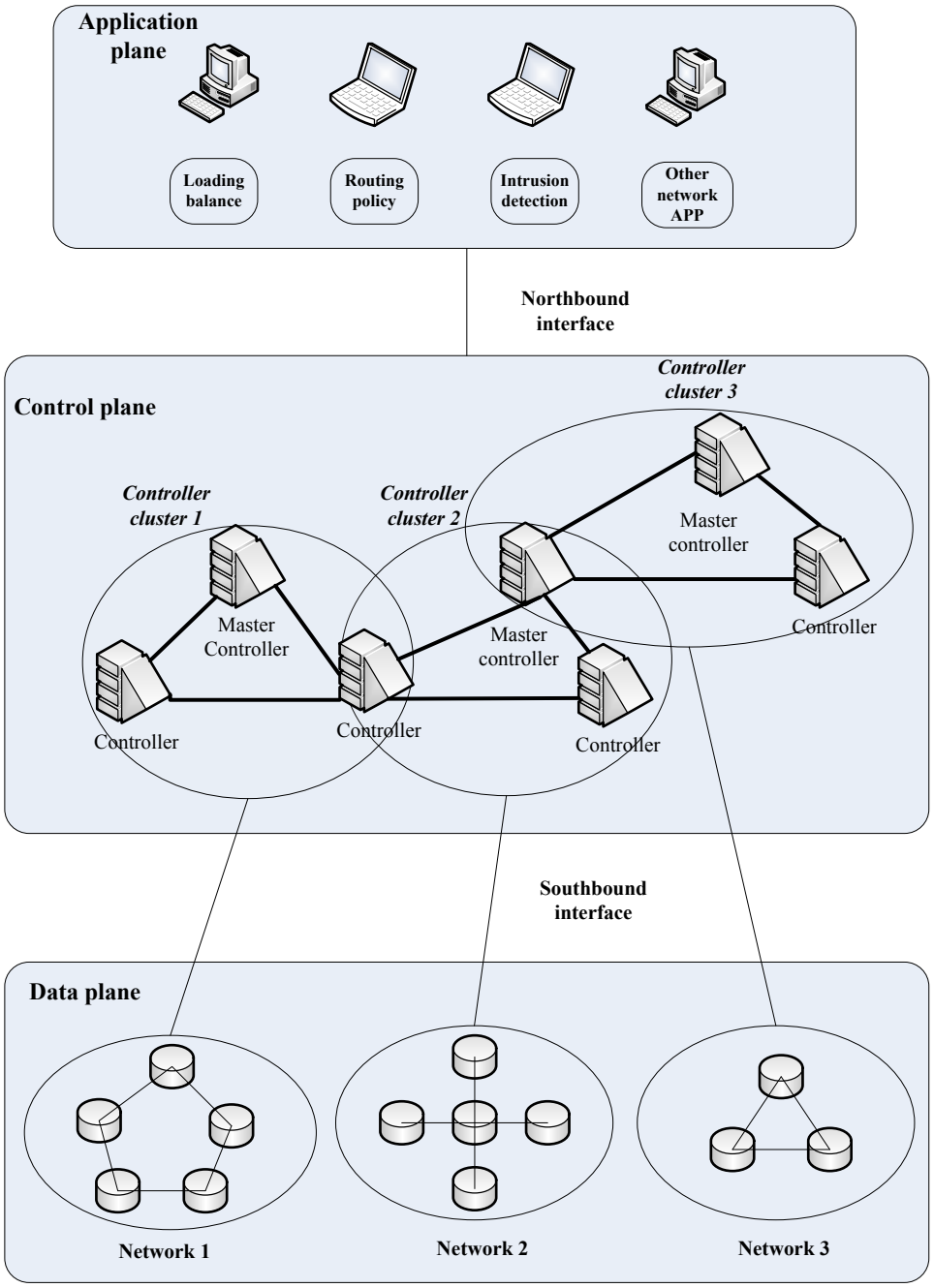

Fig.1. The architecture of SDN

As shown in Figure 1, the mainstream structure [2] of SDN includes three layers: data plane, control plane and application plane. The data plane is made up of simple interconnecting switches, and these switches forward data packets based on flow tables sent from the controllers in the control plane through a southbound interface. The control plane can be described as a brain of SDN. It is mainly responsible for making routing policies, arranging the data plane's resources and maintaining network topologies. In a mature SDN architecture [13], the control plane is made up of controller clusters instead of a single controller. A controller cluster includes a master controller [35] that is mainly responsible for data plane management and some common controllers that assist the master controller to manage the network. It is

worth noting that these controller clusters are not isolated from each other. A controller can be a member of several controller clusters and a master controller may act as a common controller role in another controller cluster. This kind of design can make full use of each controller's processing power to mitigate DoS/DDoS attacks on the control plane. The application plane consists of various applications. These applications are mainly designed to achieve some specific network functions, such as loading balance, firewall, intrusion detection, network monitoring, and so on. A northbound interface is responsible for the communications between the control plane and the application plane.

Currently, Openflow [24] is the most popular communication protocol used between SDN's controllers and switches/routers. In this protocol, forwarding data packets in the data plane should follow flow tables. Figure 2 shows a structure of the flow table in Openfow version 1.5. Among them, Match Fields and Instructions are the main parts. Other parts are responsible for measurement, restriction and other auxiliary work. The structure of Match Fields is also shown in Figure 2. They are used to distinguish different kinds of data packets based on their IP addresses or other routing information. The Instructions field describes what actions should be taken for packets. These actions include required actions and optional actions. The required actions include output packets, drop packets, and set the queue of packets. The optional actions include set-Field (modify values in matching fields), change-TTL, Push-Tag/Pop-Tag and other operations.

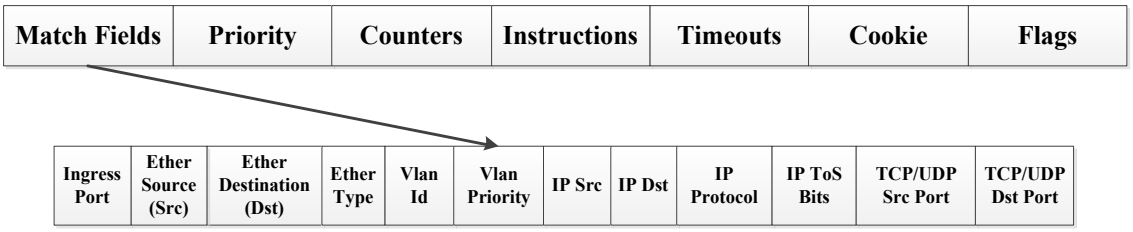

Fig.2. The structure of the flow table in Openflow version 1.5

3.2 Trusted Computing Platform (TCP)

Trusted computing is a technology to secure a computing system [25]. As we known, physical trust is the most reliable. Trusted computing can ensure that a computing system is physically trusted by applying a secure chip. By verifying a computer's hardware, operating system software, and application software step by step, trusted computing can ensure that a computing platform can always operate as expected. If the platform wants to be trusted always, it must have an ability to report its state information in the condition that its identity information and private information cannot be revealed. So, trusted computing must have the following abilities. First, it can protect the platform's sensitive information from being compromised. Second, it can measure and evaluate the integrity of the platform's entities or components based on the characteristics of the platform. Third, it can authenticate another platform's status based on its characterization.

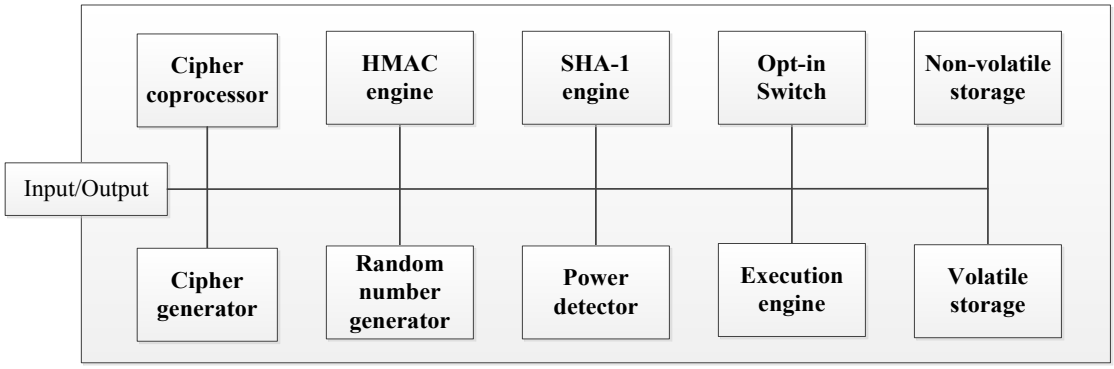

Fig.3. The structure of Trusted Computing Platform

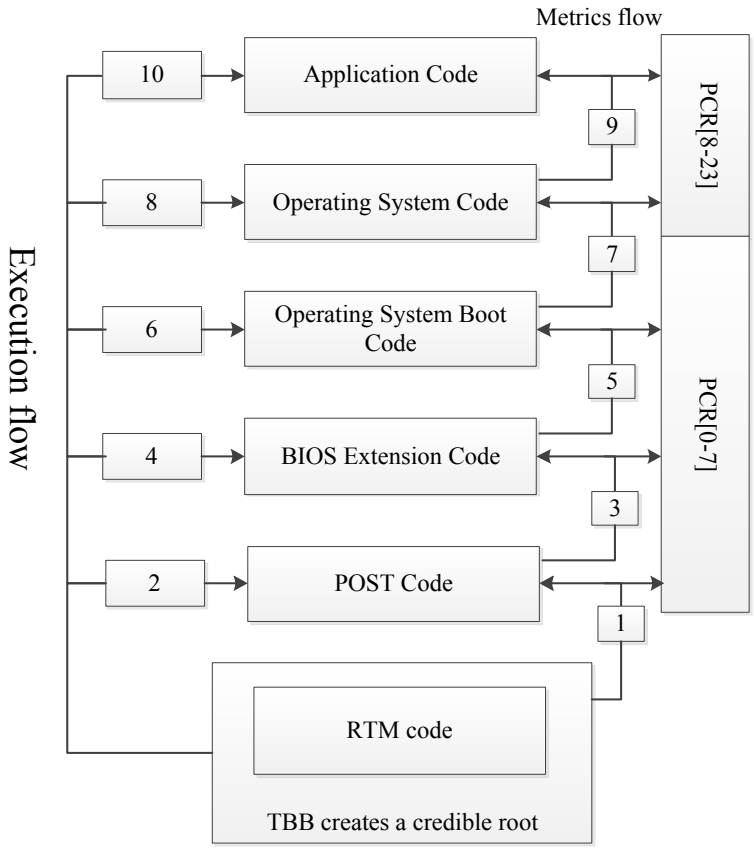

Fig.4. A trust measurement chain

In order to provide the platform security-related capabilities as described above, Trusted Computing Group (TCG) defined TCP's structure in Trusted Platform Module in TCG-1.2 [34]. As shown in Figure 3, TCP is made up of ten modules including cryptographic related components, storage modules and some system protection components that ensure TCP to work stably for a long time. Trusted computing is able to provide a secure computing platform because it uses a trust chain delivery mechanism. As illustrated in Figure 4, when a trusted platform is established, TCP's Trust Building Block (TBB) module creates a credible root to serve as trusted computing start. Then, core root trusted measurement module verifies Root Trusted Measurement (RTM) and Powers on Self-Test (POST) code in BIOS (Basic

Input/Output System). Meanwhile, the system stores the measurement summary in Platform Configuration Register (PCR). In the premise that current part and code are credible, TPM then assesses the security of next part or upcoming code and saving the measurement value into PCR. Entire startup sequence follows the principle of "first measure, then execute" [26, 27]. In this way, TCP starts with BIOS extension code, then verifies the operating system loader, finally to upper applications, and it gives each part executive power after it is proved to be credible. In TCP, a computer's all parts are verified level-by-level, the trust train is delivered from bottom to top, and it extends trust from the root to the entire platform.

TCP's measurement log stores each stage's codes, detailed measurement information of configuration status and history records of PCR values. If an attacker tampers configuration information or executes malicious codes, the log will store a new measurement value. During a platform operation process, the status information in the measurement log is provided to platform users for verification, so that the users can make a judgment on whether the platform is credible according to the credibility report that includes the measurement log and the corresponding PCR values.

4. System Design

This section presents the proposed trust management framework. Firstly, we introduce system structure and each module's functionalities. Then, we describe technical details of the TMF in terms of trust based flow rule selection, trustworthy network performance monitoring, data collection with tags and trust evaluation.

4.1 System Architecture

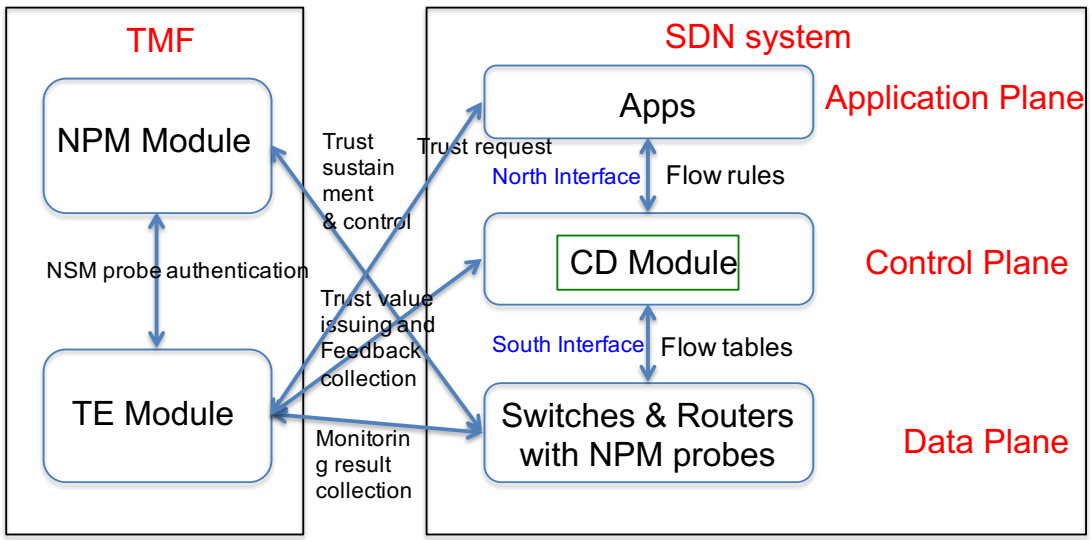

Fig.5. The structure of trust management framework

Figure 5 shows the system structure of the proposed Trust Management Framework (TMF). The SDN system is constructed with three layers: application plane, control plane and data plane. As mentioned in Section 3, network policies are generated in the

application plane and embodied in the control plane. The control plane generates data forwarding commands based on a selected flow rule to update the flow tables that are used to instruct data forwarding in the data plane. We add a Conflict Detection (CD) module in the control plane to detect whether new policy conflicts appear. The CD module decides which flow rule should be selected and applied based on the trust values of applications. The data plane mainly comprises forwarding devices such as routers and switches. Besides forwarding data flows, it also collects network data and sends network status to the control plane. The network performance data can be collected by embedding network performance monitor probes into the network devices in the data plane.

The Trust Management Framework contains two modules: Network Performance Monitor (NPM) module and Trust Evaluation (TE) module. The NPM module is composed of a number of probes that are configured to monitor the performance of flow rules issued by different applications when they are carried out and fulfilled. These probes (either hardware or software or both) are located in suitable places of the data plane in the network and are trusted by the TMF for fulfilling the network performance monitoring (by applying the trust sustainment and control mechanism based on TPM). They monitor the network performance in terms of throughput rate, packet loss probability, time delay and so on when a flow rule is applied. These monitoring results will be sent to the TE module with corresponding tags that indicate the corresponding application that issues the flow rules. By collecting the above monitoring results and also the feedback from other application users (e.g., different controllers), the TE module calculates the trust values of the applications that issue the flow rules. These trust values will be stored in TE module's database, meanwhile, they are regularly updated as long as the applications are running in the control plane. Once the trust value of an application is generated or updated based on its performance evaluation, it will be sent to the control plane's CD module in order to help it manage applications and assist it to select flow rules, even facing any conflicts caused by different applications.

We assume that any applications in the application plane can authenticate themselves with the control plane, e.g., by applying a public key cryptosystem. The data collected by the probes are signed and cannot be denied by their providers. The identity management of the system is based on Public Key Cryptosystem. The public key of the system entity represents its unique identifier. Note that TMF can be located inside the control plane or in a trusted third party and be shared by a number of control planes.

Figure 6 shows the main procedure of system process. The procedure can be described as follows:
Step 1: Before an application is installed in the application plane, the control plane checks the trust value of this application. If this application's trust value is sufficiently

high (e.g., higher than a threshold that is set 0.5), this application can be deployed. Otherwise, the application should be discarded.

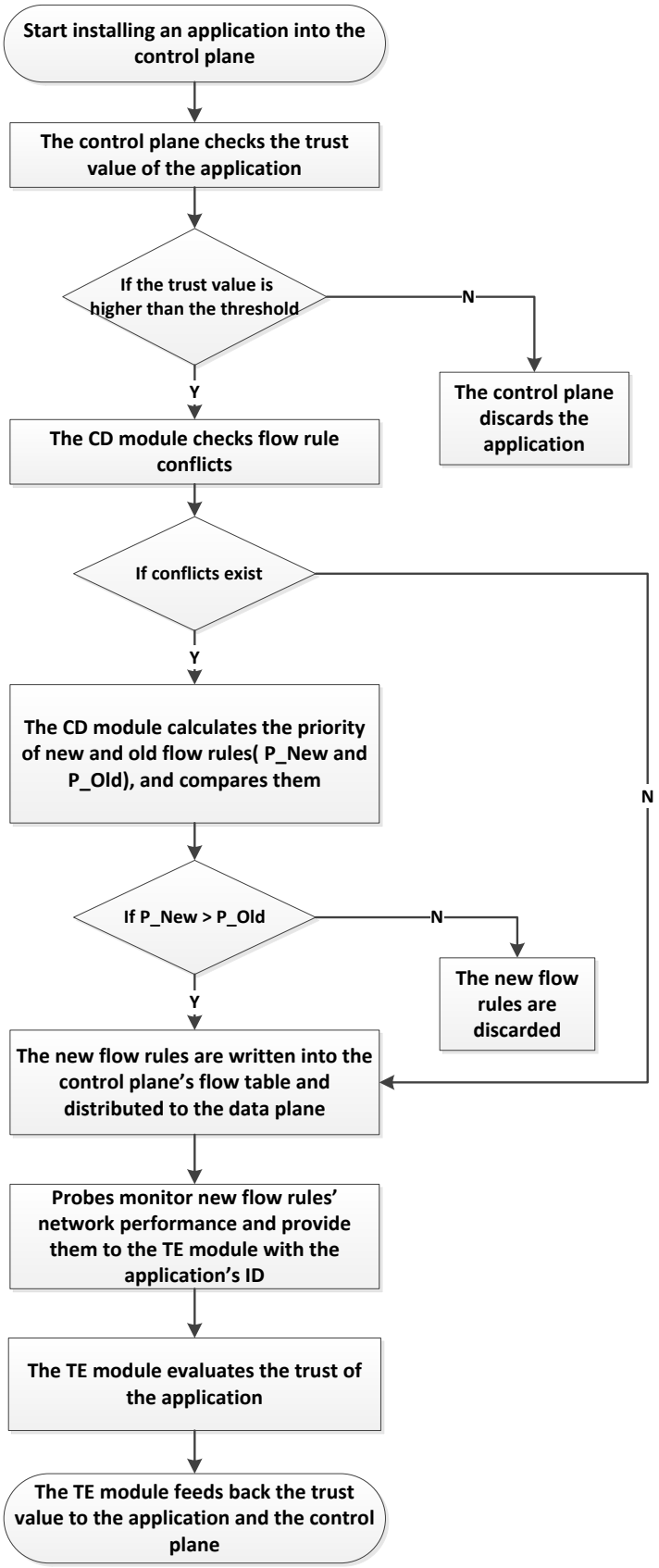

Fig.6. Trust Management Procedure

Step 2: Once an application is installed, the CD module checks if there are conflicts between new flow rules and old rules. If there are no conflicts, the new flow rules will be written into the control plane's flow table and distributed to the data plane. Otherwise, the CD module should calculate the priorities of both new and old flow rules (P_New and P_Old) and should select the flow rules with a higher priority.
Step 3: The NPM probes monitor the network performance of new flow rules after they are distributed from the control plane to the data plane and send the performance data to the TE module with the corresponding application's ID, as well as the time of collection. Then the TE module evaluates the trust value of this application and feedbacks it to the control plane for application management.

4.2 Trust Based Flow Rule Selection

Before an application provides its network requirements and desired network behaviors (i.e., network policies) to the control plane, it should first send a request to the TE module. Then the TE module checks the trust value of this application and issues this value to the application and/or the control plane. If this is a new application, the TE module gives it an initial trust value. When the application sends its policy (i.e., flow rules) to the control plane, it must sign the flow rules with its private key and attach its certified trust value (issued by the TE module, i.e., signed by the TE module). The CD module checks whether the flow rules conflicts with the existing policies. The conflict detection principle is that several flow rules have the same match field but different instructions. We define P_New represents the priority of the new flow rule provided by the new application and P_Old represents the priority of the old flow rule provided by another old application. In the process of conflict resolution, the system follows the following rules: If P_New > P_Old, the old flow rule is replaced by the new one. If P_New < P_Old, the new flow rule will be rejected. If P_New = P_Old, it will be handled by a network administrator to make a final decision or keep the old flow rule. Then the control plane converts the selected flow rules into the controller's flow table and sends them to the flow table in the data plane. In the whole process that the flow rules are carried out, the NPM module monitors the performance of their execution until they are carried out and fulfilled.

Herein, the priority of the application policy is calculated based in the trust value of the application. In a simple way, $P_i = F(T_i)$, where T_i denote application i's trust value, P_i is i's priority value. The bigger value of the T_i, the higher the P_i is. Notably, P_i could be also impacted by other factors, not only trust, thus function F could take other inputs (such as application authority level, importance level, etc.) into account in order to generate an accurate priority value of an application.

4.3 Trustworthy Network Performance Monitoring and Data Collection

In order to ensure that the Network Performance Monitor (NPM) probes can work and collect network data as expected, we apply Trusted Platform Module (TPM) [28-30] into them. TPM's working mechanism was introduced in Section 3. It can verify

platform's trustworthiness based on its hardware and operating configurations. Applying this feature, we use the NPM Module to manage the trust of NPM probes by applying the trust sustainment and control mechanism [28, 29]. Once the probe is attacked or intruded, the NPM module will be informed. Meanwhile, we also use digital signature to guarantee the trustworthiness of collected network performance data. When the monitored data is collected from a probe, the TE module checks with the NPM module to ensure that the data-providing probe is in a trusted status. Concretely, the probe sends its network performance monitoring data and its public key to the TE module by signing the data with its private key. The TE module verifies the signature and checks the trustworthiness of the probe by providing its public key to the NPM module. If the probe is in a trusted status, its provided data can be considered in the trust evaluation; otherwise, the TE module will discard the data if the probe's authentication with the NPM module fails (e.g., in a situation that the probe is attacked).

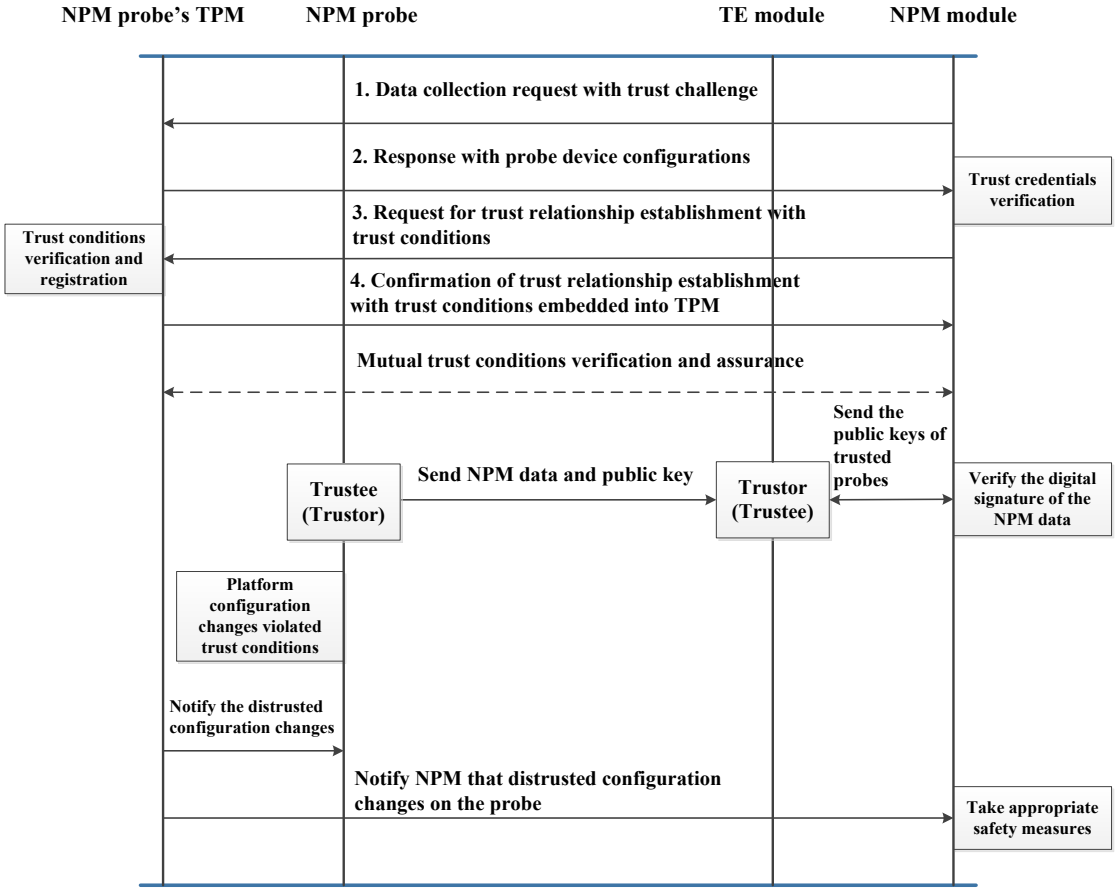

Fig.7. Procedure of trustworthy network performance monitoring

Figure 7 shows the procedure of trustworthy network performance monitoring and data collection. Herein, we apply the trust sustainment and control mechanism as described in [28, 29] based on our previous work [30]. The procedure can be described as follows:

Step 1: The NPM module sends a data collection request with a trust challenge to a

NPM probe.
Step 2: After the probe receives this message, it responses with its device configurations (trust credentials).
Step 3: Then the NPM module performs verification on trust credentials. If the verification is successful, the NPM module will request trust relationship establishment by providing its trust conditions (e.g., no platform configuration changes and no additional software allowed to install, etc.).
Step 4: When the NPM probe gets the request, it verifies and registers the trust conditions by embedding the trust conditions into its TPM. After that, it confirms the trust relationship establishment with the NPM. With this way, the trust relationship from the NPM module to the NPM probe is established.
Step 5: Optionally, the trust relationship from the NPM probe to the NPM module can be also established if needed following a similar way. If the platform configurations appear changes that violate the trust conditions, the NPM probe's TPM will notify the NPM probe and further the NPM module about the distrusted configuration changes. The NPM module will correspondingly take an appropriate safety measures, e.g., breaking down the connection with this NPM probe and informing the TE module to discard the data collected by this probe.

4.4 Tag of Data Collection

For aggregating the data collected by different probes, we need to check the tags of the data in order to aggregate the data related to a specific application for evaluating its trust. There are two ways to mark a tag on the data.

Method 1: The control plane embeds the ID of application into its flow rules that are selected and applied into the data plane. The probes know the application ID during the monitoring. But the privacy of the application could be influenced.

Method 2: The control plane informs the TE module the application IDs and the time duration of the applied flow rules. The TE module matches the flow rule selection result and its period of usage with the time of performance monitoring data collection.

In what follows, we assume that the data collected for trust evaluation has a tag attached, which indicates its corresponding application.

4.5 Trust Evaluation

There are two sources to collect data for evaluating the trust of an application: the monitored data provided by the NPM probes and the feedback from other controllers. Multiple controllers could exist in the system. The evaluation algorithm should support this situation.

$T_{i,c}$ denotes the trust value of application i at time c. It is mainly contributed by two parts: the feedback from controller j $(j = 1, \dots, J)$ at time c_j, denoted as $Tf_{i,c}$ and the performance monitoring result, denoted as $Tp_{i,c}$. First, TMF sends a request to all controllers that have interactions with application i for feedback. After receiving the

request, controller j sends back its feedback Tp_{j,i,c_j} in an honest way. This honest behavior of controllers can be assumed due to the responsibility of network operators for offering high quality networking services. $Tp_{i,c_j,j}$ denotes application i's trust value fed back from the controller j at time slot c_j. Suppose TMF receives a total of J feedback at time slot c. Then it aggregates all feedback by considering time impact (the more recent the feedback, the more valuable it is) and the value of J's impact (since the bigger the value of J, the more accurate the evaluation is) based on Formula (1):

$$Tf_{i,c} = \frac{\theta(J)}{J}\sum_{j=1}^{J} Tp_{i,c_j,j} \times e^{-\frac{|c-c_j|}{\tau}}, \qquad (1)$$

where $\theta(J) = 1 - exp\left(\frac{-J^2}{2(\sigma+\varepsilon)^2}\right)$ is the Rayleigh cumulative distribution function to model the impact of J on trust evaluation; $\varepsilon = -\frac{J}{J'}$, parameter J' is the total number of controllers in the system. Parameter τ is applied to control the decay of Tp_{j,i,c_j} since the most recent feedback should contribute more on the trust evaluation.

For calculating $Tp_{i,c}$ based on the performance monitoring results, we apply the following algorithm to prepare $Tp_{i,c}$ calculation.

Algorithm 1: Preparation of $Tp_{i,c}$ calculation

Input: Monitoring result of network throughput rate $\boldsymbol{tr}$, packet loss probability $\boldsymbol{lp}$, and time delay $\boldsymbol{td}$; the threshold value of network throughput rate $\boldsymbol{TH\_tr}$; the threshold value of packet loss probability $\boldsymbol{TH\_lp}$; the threshold value of time delay $\boldsymbol{TH\_td}$; $\boldsymbol{Fac_{tr} = Fac_{lp} = Fac_{td} = 0}$. (Note that we usually set the threshold as the value that is tested at the time when the controllers do not run any third-party applications.)

If $\boldsymbol{input = tr}$, $\boldsymbol{Fac_{tr}} = \boldsymbol{1 \Big/ 1 + e^{-x\frac{(tr-TH\_tr)}{TH\_tr}}}$;

If $\boldsymbol{input = lp}$, $\boldsymbol{Fac_{lp}} = \boldsymbol{1 \Big/ 1 + e^{-y\frac{(TH\_lp-lp)}{TH\_lp}}}$;

If $\boldsymbol{input = td}$, $\boldsymbol{Fac_{td}} = \boldsymbol{1 \Big/ 1 + e^{-z\frac{(TH\_td-td)}{TH\_td}}}$;

Where $\boldsymbol{x, y, z}$ denote the scales of the impact of $\boldsymbol{tr}$, $\boldsymbol{lp}$ and $\boldsymbol{td}$, respectively.

Output: $\boldsymbol{Fac_{tr}; Fac_{lp}; Fac_{td}}$.

After collecting all Fac from all probes at time slot c, the TE module first checks the tags attached to Fac with regard to application i $(i = 1, \dots, I)$ and then calculates $Tp_{i,c}$ by combining $Fac_{k,i}$ from probe k that contains tag i with Formula (2).

$$Tp_{i,c} = \frac{\sum_{k=1}^{K}(Fac_{tr,k,i} + Fac_{lp,k,i} + Fac_{td,k,i})}{3K + r}, \; r \geq 1, \qquad (2)$$

where r is a parameter to ensure $0 \leq Tp_{i,c} < 1$ and the validity of Formula (2). K is the total number of probes that provide monitoring results with regard to application i.

Finally, we aggregate $Tf_{i,c}$ and $Tp_{i,c}$ together and also consider past trust value $T_{i,c'}$ at a previous time slot c' based on Formula (3).

$$T_{i,c} = \alpha T_{i,c'} + \beta Tf_{i,c} + \gamma Tp_{i,c}, \qquad (3)$$

where $\alpha + \beta + \gamma = 1$, they are weighting parameters of the above three input factors.

5. System Performance Evaluation

In this section, we firstly introduce TMF prototype implementation. We then design a number of experiments to show the accuracy and efficiency of our system. At last, we analyze its robustness.

5.1 System Implementation

We implemented a TMF prototype system based on the design specified in Section 4. As shown in Figure 8, we installed floodlight version 1.2 controller in PC 1 to simulate SDN's control plane. This computer's CPU is Inter Core i3-3220 at 3.30GHZ and RAM's size is 4GB, running operating system is Ubuntu 16.04 LTS. We developed the CD module with Java and embedded it into the floodlight's source code package. The TE module, implemented with Java in PC 1, connects to the CD module to transmit trust values. The TE module consists of trust value evaluation part and MySQL database that stores applications' trust values. In PC 1, we also implemented the NPM module to support the trust sustainment and control mechanism by challenging the probe devices via TPM in order to ensure the credibility of NPM probes and the trustworthiness of network performance data collection.

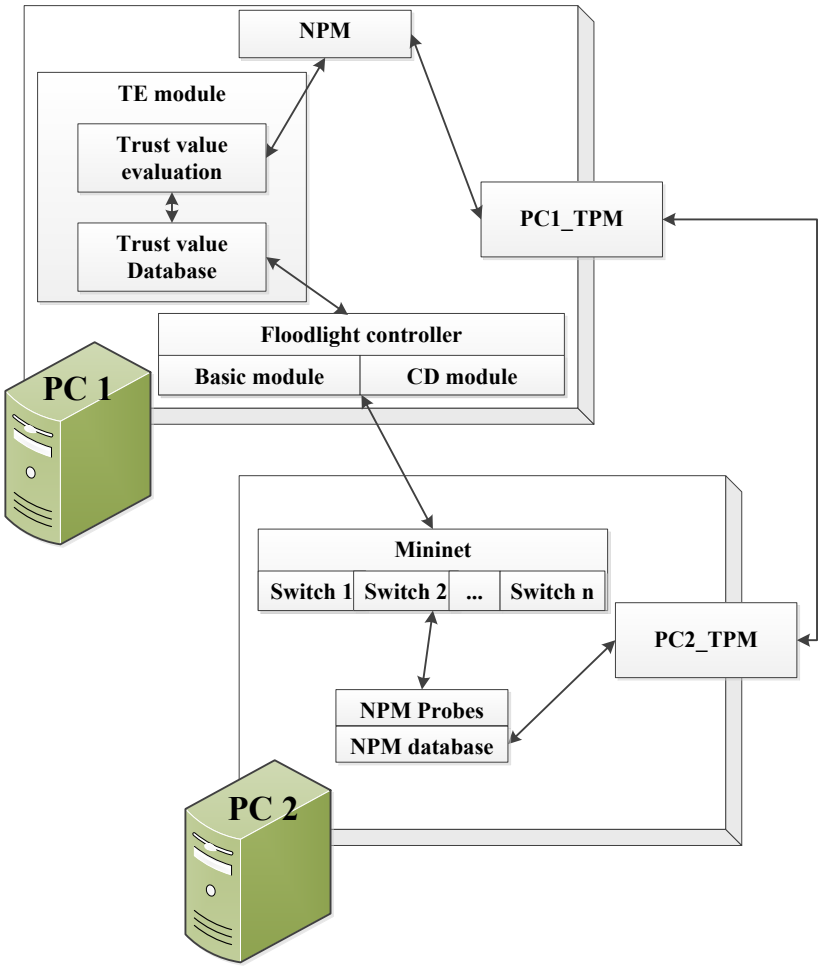

Fig.8. TMF prototype's structure

Refer to Figure 8, we installed Mininet that can create a realistic virtual network in a single machine PC 2 to simulate a number of Openflow switches. These switches make up the data plane and directly connect to the PC 1's floodlight controller. PC 2's CPU is Inter Pentinum G630 at 2.70GHZ with 4GB RAM, running operating system Red Hat Enterprise Linux Server 6.5. PC1 connects to PC 2 through a 100 Mbps wired network. We simulated NPM probes with a network measurement tool called Iperf to collect network performance data. After the data are collected, they are stored in PC 1's MySQL database. There are three tables in this database that respectively record time delay, throughput rate, and packet loss probability of network nodes affected by different applications' flow rules in different time slots. TPM is installed in both computers PC 1 and PC 2 to ensure the credibility of the NPM probes. The code in PC 2 was written in C++ language.

5.2 Performance Evaluation

Our system's main task is to evaluate the trust of applications and manage them based on it. Therefore, the accuracy of trust evaluation is the first that should be assessed. In addition, we also tested the efficiency of the prototype system in terms of CPU usage, memory usage and operation time of each system procedure.

5.2.1 Accuracy

For evaluating the accuracy of application trust evaluation, we did some preparation. Firstly, we simulated two types of network topology in the data plane. As shown in Figure 9, the first topology is a Fat-tree topology that has 20 Openflow switches and 16 hosts. This kind of network topology is usually used in data centers. The second type is a Star topology that has 6 Openflow switches and 10 hosts. Secondly, to simulate different SDN applications, we designed three loading balance applications (LB APP). In order to differentiate the performance of these three applications, we only let the first application (LB APP 1) issue the flow rules that make the transmission of data packets in network topology evenly distributed in each equivalent link. The second loading balance application (LB APP 2) generates some flow rules conflicted with the first one. Although it can also complete the task of load balancing, its performance is worse than the first one because some of its flow rules make data packets choose a further router, which results in uneven distribution of flows. The third one (LB APP 3) is an application with some malicious behaviors. In this application, there are flow rules that make some network switches regularly discard packets or delay packet forwarding. In these three applications, the first application's performance should be the best and the worst is the third one since it is malicious. Thirdly, as mentioned in Section 4, an application's trust value is mainly contributed by two parts: its performance monitoring reports from the control plane where it currently runs ($Tp_{i,c}$) and its performance feedback from other control planes (i.e., in other SDN systems) ($Tf_{i,c}$). To simulate an application's feedback from other control planes, we run the above loading balance applications in different control planes' controllers for different network topologies (i.e., SDN system 1 and SDN system 2, as shown in Figure 9). Then we calculated their $Tp_{i,c_j,j}$ and stored them in a database to serve as the feedbacks from other control planes.

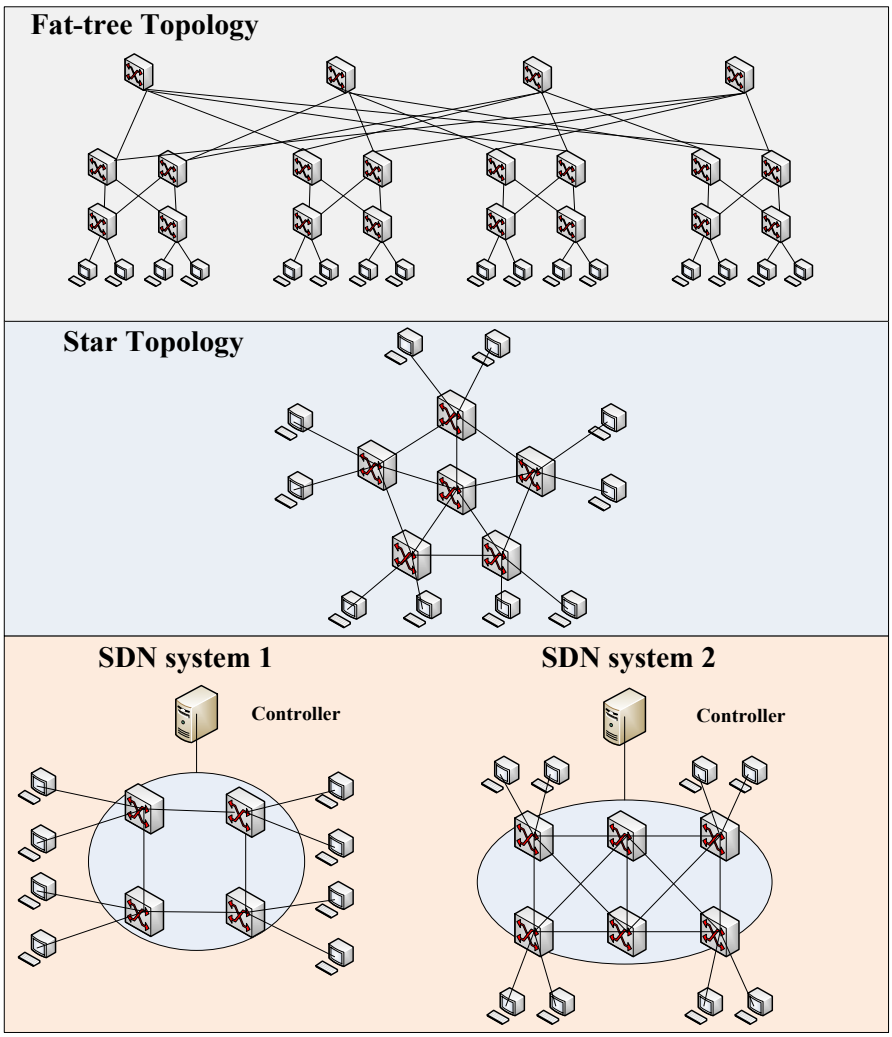

Fig.9. Network topologies applied in TMF's accuracy evaluation

Table 2. The Values of Parameters of Trust Evaluation

Formula	Parameter	Value
Formula (1)	J	2
	J'	4
	σ	3
	τ	24 hours
Algorithm 1	x	1
	y	1
	z	1
Formula (3)	α	0.2
	β	0.2
	γ	0.6

We ran the three loading balance applications in the two types of networks with Fat-tree topology and Star topology, respectively. The TMF calculated their trust

values in different time slots (every 6 hours). We set initial trust values of the three applications as 0.8. That is to say, in the trust evaluation of time slot 1, $T_{i,c'}$ of the three applications is 0.8. Table 2 lists the parameters used in our system for the calculation of trust value. In our test, J is the number of feedback pieces from other control planes, which is 2 in our test, and J' equals to the total number of control planes that is 4. In order to make the impact of feedback obtained more than one day decay faster, we set τ equal to 24 hours. We think the scales of the impact of throughput rate, packet loss probability and time delay are same, so we set parameters x, y, z as 1. In our design, an application's current performance contributes the most to its trust value, so γ was set as 0.6. The impact of the application's feedback from other control planes and its trust value in the previous time slot are smaller, compared with its impact on current networking performance. Therefore, we set both α and β as 0.2.

Table 3. Trust Values of Three Loading Balance Apps in Different Time Slots and Topologies

App Name	Time Slot 1		Time Slot 2		Time Slot 3	
	Star	Fat-tree	Star	Fat-tree	Star	Fat-tree
LB APP 1	0.823	0.815	0.827	0.816	0.828	0.816
LB APP 2	0.685	0.673	0.654	0.639	0.656	0.642
LB APP 3	0.421	0.457	0.344	0.458	0.327	0.457

The trust values of three applications are evaluated by our system and shown in Table 3. In our design, the LB APP 1 has the best quality, thus its trust value should be the highest, which is proved by our experimental result. The LB APP 3 applied some malicious policy, thus its trust value should be the lowest, which is always less than 0.5, as indicated in Table 3. The trust value of LB APP 2 falls into the middle of the trust values of LB APP 1 and LB APP 3. We can find this experimental result is the same as our expectation. In particular, when we ran the three applications in a larger topology (Fat-tree topology), we found that their performance drops slightly, compared with their performance in a Star topology. This result is reasonable since application performance drops in a larger scale network.

Table 4. Trust Values of Three Loading Balance Apps in Different Time Slots and Topologies with Different Parameters

App Name	Parameter	Time Slot 1		Time Slot 2		Time Slot 3	
		Star	Fat-tree	Star	Fat-tree	Star	Fat-tree
LB APP 1	Set 1	0.749	0.740	0.752	0.745	0.753	0.746
	Set 2	0.831	0.823	0.834	0.826	0.834	0.825
LB APP 2	Set 1	0.628	0.613	0.579	0.566	0.589	0.579

App Name	Parameter	Time Slot 1		Time Slot 2		Time Slot 3	
		Star	Fat-tree	Star	Fat-tree	Star	Fat-tree
	Set 2	0.672	0.668	0.641	0.635	0.648	0.639
LB APP 3	Set 1	0.387	0.381	0.311	0.303	0.283	0.279
	Set 2	0.382	0.376	0.342	0.336	0.334	0.328

We also evaluated the three applications' trust values with different system parameters. These trust values are listed in Table 4. Two sets of parameters sets are used to calculate the trust values. In Set 1, $\alpha = 0.2$ (the weight of previous trust value $T_{i,c'}$); $\beta = 0.3$ (the weight of feedback from other control planes $Tf_{i,c}$); $\gamma = 0.5$ (the weight of $Tp_{i,c}$), and other parameters are the same as the parameters in Table 2. In Set 2, $\alpha = 0.1$; $\beta = 0.2$; $\gamma = 0.7$, and other parameters are same as the parameters in Table 2. We can see the weighting parameters' impact on the trust evaluation.

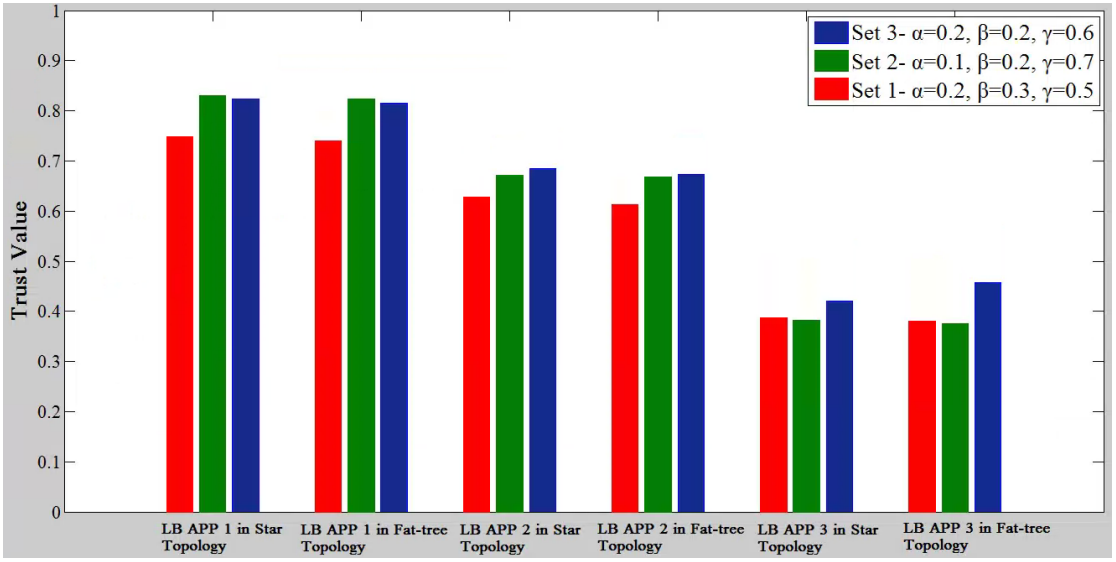

Fig.10. Three LB APPs' trust values in time slot 1 in different networks and with different parameters

We further compared the three applications' trust values in time slot 1 with different parameter sets (Set 1, Set 2, and parameters in Table 2, denoted as Set 3), as shown in Figure 10. Comparing the results with Set 1 and Set 3, we can find that the three applications' trust values become big when α remains the same, β decreases and γ increases. This is because in our design, the number of pieces of feedback from other control planes is relatively small, resulting in $Tf_{i,c}$ is smaller than $Tp_{i,c}$, which ultimately lead the applications' trust values increase when β decreases. The purpose of comparison between Set 2 and Set 3 is to observe the influence of α and γ when β stays the same. As mentioned before, $T_{i,c'}$ in Time Slot 1 equals 0.8, it's smaller than LB APP 1's $Tp_{i,c}$ and bigger than LB APP 2 and 3's $Tp_{i,c}$. Therefore, the trust value of LB APP 1 evaluated in Set 3 is smaller than in Set 2, and the evaluation of trust values of LB APP 2 and APP 3 shows opposite results. To sum up, our system

can objectively evaluate an application's trust by adjusting the weight of each parameter in various scenarios. For example, when we don't need to pay much attention to the feedback from other control planes, we can reduce the value of β to keep the accuracy of trust values.

5.3.2 Efficiency

In system efficiency test, we measured the floodlight controller's CPU usage, memory usage and each procedure's operation time. Firstly, we ran floodlight version 1.2 without the TMF support and connected the floodlight controller to the network with the Fat-tree topology. Then we installed the three loading balance applications separately and record the controller's CPU usage and memory usage. We also did the same experiment in the floodlight controller with the TMF support and recorded the related data. The test results are presented in Figure 11 and 12, respectively.

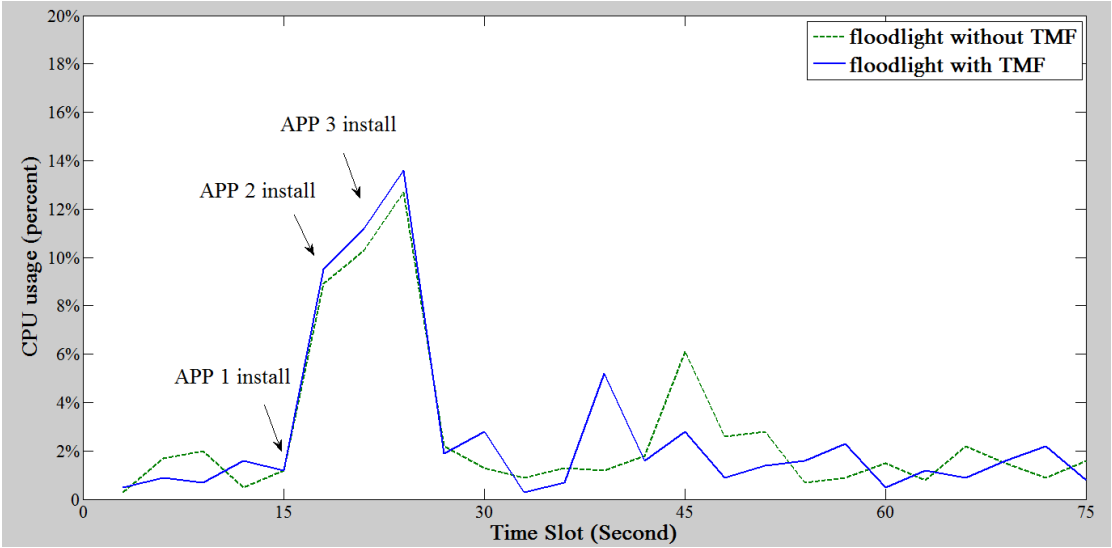

Fig. 11. CPU usage of floodlight version 1.2 with and without TMF

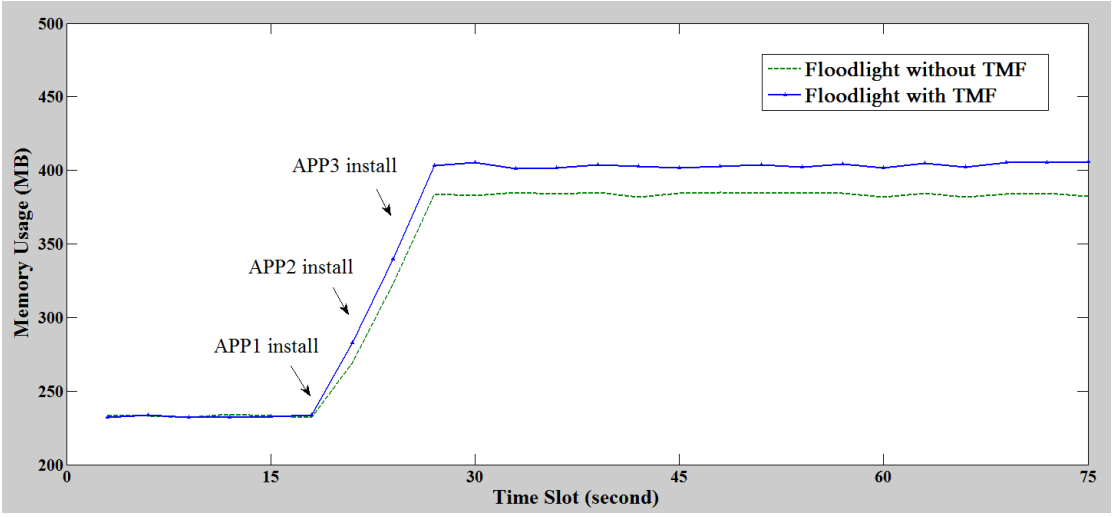

Fig. 12. Memory usage of floodlight version 1.2 with and without TMF

As shown in Figure 11 and Figure 12, when a new application is installed into the

controller, its CPU and memory usage will increase because the controller need to load the application and insert new rules into its flow table. When running the TMF, more resources are consumed, because the controller needs to check the application's trust value, detect flow rule conflicts and select the flow rules of the application with the highest trust value. Through comparison, we can see that the CPU consumption of the TMF is acceptable. The memory usage of the controller with the TMF support is about 5% more than that without TMF. In short, the TMF consumes few resources for achieving desired functionalities.

Table 5. The Operation Time of TMF (unit: millisecond)

Procedure	**Operation Time**		
	APP 1	**APP 2**	**APP 3**
CD module	22	21	23
TPM verification	954	936	973
NPM data transmission and verification	42	46	45
Trust Evaluation	5	5	5
Trust value feedback	10	12	9

To test the operation time of TMF, we successively installed three load balancing applications that generate some flow rules conflicting with each other into the controller with Fat-tree topology. After that, we separately recorded the operation time of TMF's each procedure. The result is shown in Table 5. In each procedure, the trusted platform module spends the most time to do platform verification, but this operation only needs to be done few times when NPM probes establish connection with the NPM module and when the NPM module needs to perform periodical re-verification via TPM. In this test, each application generated about 500 flow rules, so the CD module detected about five hundred flow rule conflicts and made selection each time. The operation time of conflict detection is less than 25 milliseconds. The operation time of trust value feedback and distribution is about 10 milliseconds. Trust value evaluation spends the least time, which is less than 10 milliseconds. In general, the operation time of the TMF is relatively low. Taking both the operation time and the resource consumption into account, we can see that the TMF system is efficient and can provide precise trust management for SDN applications based on digital trust evaluation.

In what follows, we further compared our system's performance with some related work [15, 16, 19] reviewed in Section 2.

1) We compared the operation time of adding new flow rules into the controller with the existing works in [15, 16]. They proposed role-based solutions to deal with flow rule conflicts. By installing different numbers of applications, we inserted

different number of flow rules into the controller's flow table. After that, we tested the operation time of adding different number of flow rules. The result is shown in Figure 13. By contrast, we find that our system has lower operation time than existing work when detecting the same number of flow rule conflicts.

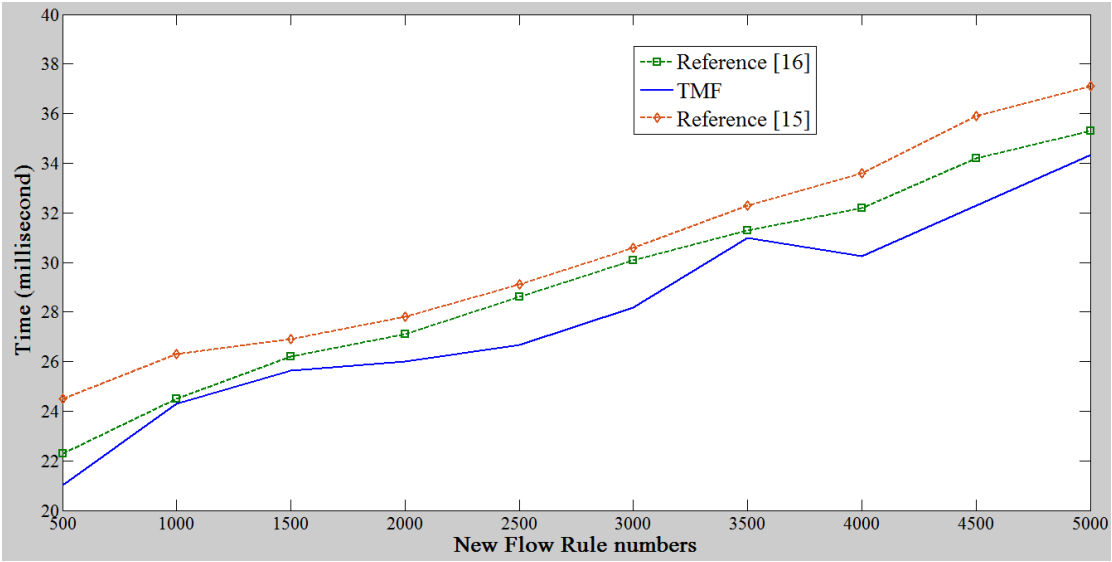

Fig. 13. Comparison of operation time to add new flows

2) We tested the time of installing a new application into the controller with the existing work [19]. In [19], a permission check is performed before each application is installed into the controller. We installed an application with 50 flow rules into the controller for ten times, recorded their average time and calculated standard deviation of them to show their volatility. The result is shown in Figure 14. We find that the time of our system to add a new application is similar to that of [19], but our system has smaller standard deviation.

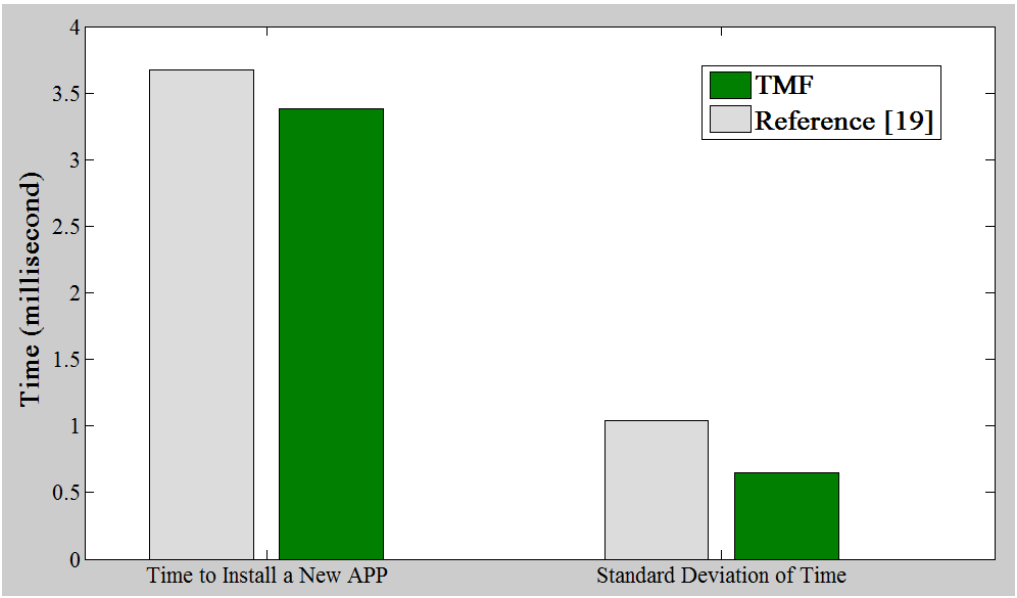

Fig. 14. Comparison of time to install a new application

It is not easy to compare our work with other existing ones, we present the reasons below. The system main functions in [17], [18] and [22] are different from ours, so the objectives of system performance evaluation are different from ours. In [20], [21] and [22], authors only showed their system design without any system performance evaluation.

5.3 System Robustness Analysis

The core function of our system is trust evaluation based on the network performance data collected by the NPM probes. In order to ensure the system core function to run correctly, we must guarantee the veracity of the collected data. As described in Section 4, we embedded TPM into the NPM probes, applied a trust sustainment and control mechanism [28, 29] to ensure the truth of network performance data collection and used digital signature to achieve non-repudiation. The TPM module can ensure the NPM probes to perform as expectation for collecting desired network performance data. Once a NPM probe is compromised or behaves maliciously, its configuration information is changed. Meanwhile, the probe's TPM will inform the NPM module that this probe is beyond credible, and the NPM module will break down the connection between TE module and this probe. The TE module will not use the data collected from this probe for trust evaluation. To prevent the collected network performance data from being tampered, we used digital signature in our prototype system. Concretely, we used SHA-256 hash function to extract message digest of data and RSA asymmetric cryptography to sign the extracted message digest. So, once the network performance data is tampered by attackers, the NPM module can find that its message digest is changed, and then discard the data. If this event happens many times, the NPM module will inform network administrators and the TE module to disconnect with the related probes. Based on the above analysis, we can see that our system has certain resistance to attacks against the NPM probes.

In our system, before the TE module evaluates an application's trust value, it should collect a large amount of network performance data through the NPM probes. After an application's trust value is generated, the network performance data is useless. But the network performance data could be a target of attackers. If the TE module is attacked, these data will leak network privacy. In order to mitigate this issue, once an application's trust value is generated, its network performance data will be deleted by the TE module if they are useless. Particularly, we can encrypt useful data and store them in a backup server for later tracking.

6. Conclusion

In this paper, we proposed a trust management framework for SDN applications. It can evaluate trust of an application that is adopted by the SDN controller based on its performance and can also update the evaluated trust value dynamically accordingly to newly collected network performance data by a number of NPM probes. Besides that, the proposed TMF can detect flow rule conflicts between applications. Trust values obtained by TMF can be used to solve flow rule conflicts and detect malicious

applications. In addition, the TMF also ensure the robustness of trust evaluation by adopting a trust sustainment and control mechanism for network performance data collection, which can verify whether a network device is working in a desired manner. We implemented a prototype system to demonstrate the TMF in order to assess its performance in terms of accuracy and efficiency. We also compared the performance of our system with the existing work. The testing results are satisfactory. However, this is a simulation test. In the future, we will establish our framework and evaluate its performance in a real and large SDN environment.

Acknowledgement

This work is sponsored by the National Key Research and Development Program of China (grant 2016YFB0800704), the NSFC (grants 61672410 and U1536202), the Academy of Finland (grant 308087), the Project Supported by Natural Science Basic Research Plan in Shaanxi Province of China (Program No. 2016ZDJC-06), the Fundamental Research Funds for the Central Universities (JBG161509), and the 111 project (grants B16037 and B08038). The corresponding author is Zheng Yan.

References

[1] Matias J, Garay J, Toledo N, Unzilla J, Jacob E. Toward an SDN-enabled NFV architecture. *IEEE Communications Magazine* 2015; **53** (4): 187-193. DOI: 10.1109/MCOM.2015.7081093.

[2] Kreutz D, Ramos F M, Verissimo P E, Rothenberg C E, Azodolmolky S, Uhlig S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE* 2015; **103** (1): 14-76. DOI: 10.1109/JPROC.2014.2371999.

[3] H Kim, N Feamster. Improving network management with software defined networking. *IEEE Communications Magazine* 2013; **51** (2): 114-119. DOI: 10.1109/MCOM.2013.6461195.

[4] Trivisonno R, Guerzoni R, Vaishnavi I, Soldani D. SDN‐based 5G mobile networks: architecture, functions, procedures and backward compatibility. *Transactions on Emerging Telecommunications Technologies* 2015; **26** (1): 82-92. DOI: 10.1002/ett.2915.

[5] Scott-Hayward S, Natarajan S, Sezer S. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials* 2016; **18** (1): 623-654. DOI: 10.1109/COMST.2015.2453114.

[6] Yan Q, Yu F R, Gong Q, Li, J. Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys & Tutorials* 2016; **18** (1): 602-622. DOI: 10.1109/COMST.2015.2487361.

[7] Ali S T, Sivaraman V, Radford A, Jha S. A survey of securing networks using software defined networking. *IEEE transactions on reliability* 2015; **64** (3): 1086-1097. DOI: 10.1109/TR.2015.2421391.

[8] Lim S, Ha J, Kim H, Kim Y, Yang S. A SDN-oriented DDoS blocking scheme for botnet-based attacks. *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on*, July 2014. IEEE; 63-68. DOI: 10.1109/ICUFN.2014.

6876752.

[9] Mousavi S M, St-Hilaire M. Early detection of DDoS attacks against SDN controllers. *Computing, Networking and Communications (ICNC), 2015 International Conference on*, February 2015. IEEE; 77-81. DOI: 10.1109/ICCNC.2015.7069319.

[10] Braga R, Mota E, Passito A. Lightweight DDoS flooding attack detection using NOX/OpenFlow. *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, October 2010. IEEE; 408-415. DOI: 10.1109/LCN.2010.5735752.

[11] Oktian Y E, Lee S, Lee H. Mitigating denial of service (dos) attacks in openflow networks. *Information and Communication Technology Convergence (ICTC), 2014 International Conference on*, October 2014. IEEE; 325-330. DOI: 10.1109/ICTC.2014.6983147.

[12] Lim S, Yang S, Kim Y, Yang S, Kim H. Controller scheduling for continued SDN operation under DDoS attacks. *Electronics Letters*, 2015; **51** (16): 1259-1261. DOI: 10.1049/el.2015.0334.

[13] Dabbagh M, Hamdaoui B, Guizani M, Rayes A. Software-defined networking security: pros and cons. *IEEE Communications Magazine* 2015; **53** (6): 73-79. DOI: 10.1109/MCOM.2015.7120048.

[14] Belyaev M, Gaivoronski S. Towards load balancing in SDN-networks during DDoS-attacks. *Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 First International*, October 2014. IEEE; 1-6. DOI: 10.1109/MoNeTeC.2014.6995578.

[15] Porras P, Shin S, Yegneswaran V, Fong M, Tyson M, Gu G. A security enforcement kernel for OpenFlow networks. *Proceedings of the first workshop on Hot topics in software defined networks*, August 2012. ACM; 121-126. DOI: 10.1145/2342441.2342466.

[16] Porras P A, Cheung S, Fong M W, Skinner K, Yegneswaran V. Securing the Software Defined Network Control Layer. *The Network and Distributed System Security Symposium*, February 2015.

[17] Shin S, Porras P A, Yegneswaran V, Fong M W, Gu G, Tyson M. (2013, February). FRESCO: Modular Composable Security Services for Software-Defined Networks. *ISOC Network and Distributed System Security Symposium (NDSS)*, February 2013.

[18] Ferguson A D, Guha A, Liang C, Fonseca R, Krishnamurthi S. Participatory networking: An API for application control of SDNs. *ACM SIGCOMM computer communication review* 2013; **43** (4): 327-338. DOI: 10.1145/2534169.2486003.

[19] Scott-Hayward S, Kane C, Sezer S. Operationcheckpoint: Sdn application control. *In Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, October 2014. IEEE; 618-623. DOI: 10.1109/ICNP.2014.98.

[20] Banse C, Rangarajan S. A secure northbound interface for sdn applications. *Trustcom/BigDataSE/ISPA,* 2015; **1**: 834-839. DOI: 10.1109/Trustcom.2015.454.

[21] Chandrasekaran B, Benson T. Tolerating SDN application failures with LegoSDN. *In Proceedings of the 13th ACM workshop on hot topics in networks*, October 2014. ACM; 22-28. DOI: 10.1145/2670518.2673880.

[22] Shin S, Song Y, Lee T, Lee S, Chung J, Porras P, Kang B B. Rosemary: A robust, secure, and high-performance network operating system. *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, November 2014. ACM; 78-89. DOI: 10.1145/2660267.2660353.

[23] Wen X, Chen Y, Hu C, Shi C, Wang Y. Towards a secure controller platform for openflow applications. *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, August 2013. ACM; 171-172. DOI: 10.1145/2491185.2491212.

[24] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 2008; **38** (2): 69-74. DOI: 10.1145/1355734.1355746.

[25] Yan Z. Trust management in mobile environments–usable and autonomic models. *IGI Global 1*, 2013; (1) 2.

[26] Ren J, Liu L, Zhang D, Zhang Q, Ba H. Tenants attested trusted cloud service. *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, June 2016. IEEE; 600-607. DOI: 10.1109/CLOUD.2016.0085.

[27] Lauer H, Kuntze N. Hypervisor-based attestation of virtual environments. *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People and Smart World Congress, 2016 Intl IEEE Conferences*, July 2016.IEEE; 333-340.DOI:10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2016.0067.

[28] Yan Z, Zhang P, Vasilakos A V. A security and trust framework for virtualized networks and software‐defined networking. *Security and communication networks* 2016; **9** (16): 3059-3069. DOI: 10.1002/sec.1243.

[29] Yan Z, Cofta P. A mechanism for trust sustainability among trusted computing platforms. *International Conference on Trust, Privacy and Security in Digital Business*, August 2004. Springer, Berlin, Heidelberg. TrustBus 2004: Trust and Privacy in Digital Business; 11-19. DOI: 10.1007/978-3-540-30079-3_2.

[30] Xu G W, Tang Y K, Yan Z, Zhang P. TIM: A trust insurance mechanism for network function virtualization based on trusted computing. *The 10th International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage (SpaCCS2017)*, GuangZhou, China, December 2017. Springer; 139-152. DOI: 10.1007/978-3-319-72389-1_13.

[31] Yao Z, Yan Z. Security in Software-Defined-Network: A Survey. *The 9th International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage (SpaCCS 2016)*, Zhangjiajie, China, November 2016. LNCS, Springer; 319-332. DOI: 10.1007/978-3-319-49148-6_27.

[32] Bian S S, Zhang P, Yan Z. A Survey on Software-Defined Networking Security. *IW5GS 2016 held in conjunction with 9th EAI International Conference on Mobile Multimedia Communications, MobiMedia2016*, Xi'an, China, June 2016.

[33] Fu Y, Yan Z, Li H, Xin X L, Cao J. A secure SDN based multi-RANs architecture for future 5G networks. *Computers & Security*, 2017; **70**: 648-662. DOI: 10.1016/j.cose.2017.08.013.

[34] Trusted Computing Group (TCG), TPM Main Specification, Version 1.2 Revision

94. https://trustedcomputinggroup.org/tpm-1-2-protection-profile/ [14 July 2014].
[35] Suh D, Pack S. Low-Complexity Master Controller Assignment in Distributed SDN Controller Environments. *IEEE Communications Letters*, 2017. DOI: 10.1109/LCOMM.2017.2787590.
[36] Wang M, Liu J, Chen J, Liu X, Mao J. Perm-guard: Authenticating the validity of flow rules in software defined networking. *Journal of Signal Processing Systems*, 2017; **86** (2-3): 157-173. DOI 10.1007/s11265-016-1115-8.
[37] Moyano R F, Cambronero D F, Triana L B. A user-centric SDN management architecture for NFV-based residential networks. *Computer Standards & Interfaces*, 2017; **54** (4): 279-292. DOI: 10.1016/j.csi.2017.01.010.
[38] Costa-Perez X, Garcia-Saavedra A, Li X, Deiss T, De La Oliva A, Di Giglio A, Moored A. 5G-Crosshaul: an SDN/NFV integrated fronthaul/backhaul transport network architecture. *IEEE Wireless Communications*, 2017; **24** (1): 38-45. DOI: 10.1109/MWC.2017.1600181WC.
[39] Wu A, Liu R, Ni W, Kaafar D, Huang X. AC-PROT: An Access Control Model to Improve Software-Defined Networking Security. *IEEE 85th Vehicular Technology Conference*, April 2017. DOI: 10.1109/VTCSpring.2017.8108543.
[40] He B, Dong L, Xu T, Fei S, Zhang H, Wang W. (2017). Research on network programming language and policy conflicts for SDN. *Concurrency and Computation: Practice and Experience*, 2017; **29** (19). DOI: 10.1002/cpe.4218.
[41] Ma Y, Wu Y, Ge J, Li J. An Architecture for Accountable Anonymous Access in the Internet-of-Things Network. *IEEE Access*, 2018. DOI: 10.1109/ACCESS. 2018.2806483.
[42] Huang C, Min G, Wu Y, Ying Y, Pei K, Xiang Z. Time Series Anomaly Detection for Trustworthy Services in Cloud Computing Systems. *IEEE Transactions on Big Data*, 2017. DOI: 10.1109/TBDATA.2017.2711039.